FIRMIN Thomas
DESCAMPS Simon

Tuteur : Bernard Carré

Rapport Final :

SD-Graphes

Planification de livraison

Année universitaire 2018-2019

Sommaire :

1. Introduction:

Le but du projet est de résoudre des problèmes de livraisons entre un dépôt et des clients à l'aide d'une modélisation par les graphes. Pour cela il faut trouver la planification optimale de minimum. Le coût ici est la distance parcourue par l'ensemble des camions. Pour résoudre ce problème nous utiliserons la méthode route-first/cluster-second qui sera expliquée plus tard. Cette méthode est un heuristique, nous obtiendrons donc une solution s'approchant de la solution optimale.

2. Cahier des charges

Il y a Cn clients à livrer, et il n'y a pas de limite de véhicules. La modélisation du problème se fera par le graphe suivant:

- Les sommets représenteront le dépôt (D) et les clients
- Les arêtes représenteront les routes entre tous les clients et avec le dépôt.

Il s'agit donc d'un graphe complet et simple, tous les sommets sont reliés entre eux, et il n'y a pas d'arête parallèle ou de boucle.

Selon le fichier de données fourni un client est relié à tous les autres, et le dépôt est relié à tous les clients. Nous utilisons ici des arêtes car les routes sont à double-sens (toujours selon les données fournies par le fichier).

Il n'y a pas besoin de modéliser les camions par des sommets, car l'algorithme SPLIT gère le nombre de camion, et utilise seulement le Tour géant, composé du chemin entre les clients.
Il n'y a pas besoin de modéliser le problème, par un problème de flots, car la demande d'un client ne peut pas excéder la capacité d'un camion. En effet l'algorithme SPLIT charge les camions avec la demande totale du client, si il reste de la place dans le camion, l'algorithme tente de le charger avec la demande totale du client suivant. Sinon il prend un nouveau camion. Si jamais la demande totale d'un des clients excède la capacité du camion(Q), alors il est impossible de créer un arc du sous-graphe H, et nous nous retrouvons bloqué.

De ce fait puisque le nombre de camions est infini et que l'algorithme SPLIT tente de satisfaire la demande de tous les clients, il n'aurait pas été judicieux d'utiliser des flots.

3. Structure de Données utilisées:

- **Graphe initial:**

Le graphe initial sera modélisé par une matrice d'adjacence particulière. Lorsqu'il y a une arête entre 2 sommets, plutôt que de mettre VRAI (ou 1), on insère le poid de l'arête reliant les 2 sommets. Une modélisation du type Head/Succ n'aurait pas était judicieuse car nous aurions eu besoin d'un 3ème tableau avec les poids de chaque arête. (m: arêtes, n:sommets)

Taille matrice = (n*n)=n²

Taille tableaux Head+Succ+Poids = n+m+m (graphe complet m = n(n-1)/2)

= n²

Mais puisqu'il s'agit d'un graphe complet l'utilisation de la matrice d'adjacence décrite ci-dessus reste plus simple d'utilisation et plus compréhensible.

- **Algorithme du Tour Géant:**

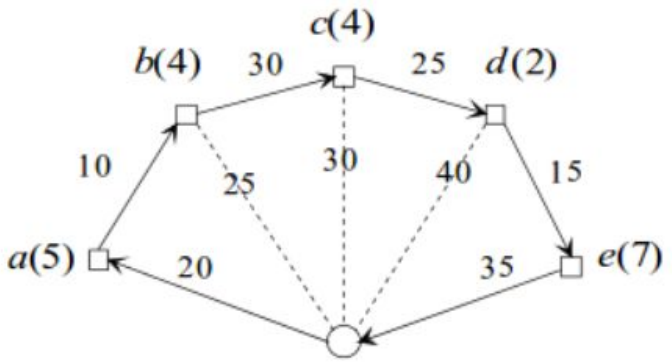
L'algorithme du Tour géant permet de créer un chemin Hamiltonien (chemin élémentaire passant une et une seule fois par tous les sommets excepté D)

De ce fait il est préférable d'utiliser un tableau de sommets comme résultat de cet algorithme. En effet le parcours de ce chemin est donc très simple.

Utiliser des tableau Head/Succ est inutile. Puisque l'indice pointé par Head est aussi le numéro du sommet décrit par les indices de parcours de Head, on perd ainsi tout l'intérêt de cette structure. Cela revient donc à utiliser un tableau de taille n sommets.
Dans les 2 cas il n'y a pas besoin de stocker les poids des arcs puisqu'il n'existe qu'une seule arête entre 2 sommets. Comme l'algorithme SPLIT a besoin du Tour géant et des distances entre tous les sommets, il est inutile de stocker les poids des arcs du Tour Géant.

Exemple pour le graphe 1:

Head/Succ	Tableau de sommets
Head = [1 || 2] Succ = [2 || 3]	T = [1,2,3]
Le sommet i a pour successeur le sommet d'indice Succ[Head[i]], sauf que Head[i]=i. Le tableau Head devient ainsi inutile. On pourrait donc, simplement utiliser Succ : Le sommet i a pour successeur Succ[i].	Le sommet T[i] a pour successeur T[i+1], si T[i] n'a pas de successeur alors le chemin est terminé.

<table>
<tr>
<td> 
</td>
<td>
- Si on choisit le point a en point de départ, le coût total après l'algorithme du tour géant sera de 20 + 10 + 30 + 25 + 15 + 35 = 135
- Si on choisit le point b en point de départ, le coût total sera de 25 + 10 + 15 + 25 + 15 + 35 = 125

Nous serons donc amenés à nous demander quel point choisir au départ dans un soucis d'optimalité
</td>
</tr>
</table>

- **Algorithme SPLIT:**

L'algorithme SPLIT prend en paramètre les structures particulières suivantes:

- le tour géant (tableau de sommet)
- les distances entre tous les sommets (Matrice d'adjacence avec le poid des arêtes).

Et renvoie un sous-graphe H.

Ce sous-graphe pourra être modélisé cette fois par un tableau de listes chaînées.
En effet chaque élément du tableau contiendra le sommet h(i) ainsi que la liste de ses successeurs éventuels, accompagné des poids des arcs reliant h(i) aux successeurs.
On obtient donc une table de liste, dont les sommets de tête de chaque liste (h(i)) sont triés par ordre topologique.

Utiliser une représentation Head/Succ, implique d'implémenter un nouveau tableau de la taille de Head, contenant les poids de chaque arc entre les sommets de Head et leurs successeurs.

<table>
<tr><th>Tableau de listes</th><th>Head/Succ/Poid</th><th>Matrice</th></tr>
<tr><td>(sommet,cout)
->: pointe vers

H[0]:(1,20)->(2,5)->.....
.
.
.
H[i]:(80,32)->(25,50)->....</td><td>Head = [1 || 3|| 6]
Succ = [1,2 || 3,4,5||6,7,8]
Poid = [10,20,5,8,36,78,3]</td><td><table><tr><th>sommet</th><th>0</th><th>1</th><th>2</th></tr><tr><td>0</td><td>0</td><td>20</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td><td>30</td></tr><tr><td>2</td><td>60</td><td>0</td><td>0</td></tr></table></td></tr>
<tr><td>Permet de minimiser l'espace mémoire utilisé et facilite le parcours et l'insertion des successeurs du sommet i.</td><td>Parcours et insertions des successeurs et des poids complexes</td><td>L'espace mémoire n'est pas optimisé.
On considère que M[i][j] correspond au poids de l'arc entre i et j; ce poid vaut 0 quand l'arc n'existe pas.
La matrice est donc creuse. beaucoup de valeurs à 0 et inutiles.</td></tr>
</table>

4. Précisions sur les algorithmes utilisés

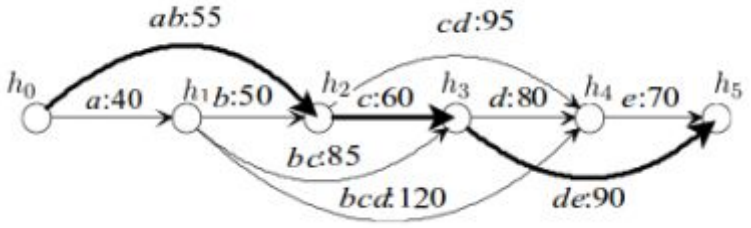
- Tour Géant:

L'algorithme fonctionne ainsi: A partir d'un premier sommet, on recherche l'arc de poids minimum vers un successeurs non-exploré de ce premier sommet. On insère le premier sommet et sont successeur dans le résultat. On considère ensuite le successeur. On rechercher l'arc de poids minimal vers un successeur non-exploré de ce sommet. On insère ce nouveau sommet dans le résultat. Et ainsi de suite jusqu'à ce que la taille du résultat soit égal au nombre total de sommet.

- SPLIT

Le graphe auxiliaire H correspond aux différentes manières de résoudre le problème en considérant le tour géant. Les sommets représentent les étapes de livraison et les arcs le coût pour parvenir à cette étape. Chaque étape correspond à un nombre de clients livré. L'ordre de ces étapes est défini par le Tour Géant. L'étape 0 est le point de départ (le dépôt) et l'étape n est le sommet n du tour Géant. Chaque arc correspond donc à l'utilisation d'un nouveau camion et le coût lié à la satisfaction de l'étape par ce camion.

Prenons l'exemple donnée:



Du dépôt (étape 0) nous pouvons aller soit à l'étape 1 soit à l'étape 2. Si l'on va à l'étape 1, seul a est livré pour un coût de 40 par 1 camion. Si l'on va à l'étape 2, a et b sont livré par un même camion pour un coût de 55. Alors qui si nous étions allé à l'étape 2 en passant par l'étape 1, nous aurions eu besoin de 2 camions, puisqu'il y a 2 arcs séparant l'étape 0 et l'étape 2.

Le but est donc de minimiser le nombre d'arcs et le poid pour aller de l'étape de livraison 0 à l'étape n.

- Plus court chemin (PCC)

Le graphe que nous obtenons par SPLIT est un graphe orienté, sans circuit, dont les poids des arcs sont positifs. Et les sommets sont par convention ordonnée par niveau, selon l'algorithme SPLIT et la structure que nous utilisons (tableau de listes chaînées).

En effet on ne peut pas passer d'une étape de livraison i à une étape de livraison i-1. Puisqu'être à l'étape i signifie que toutes les étapes précédentes sont satisfaites (tous les clients jusqu'à cette étape sont livrés). Revenir en arrière signifie que le camion revient vers un client déjà livré. Comme le sous-graphe initial des clients est complet, il y a donc n niveaux.

Ainsi il est plus judicieux d'utiliser Bellman, car avec notre structure et SPLIT, on s'abstient du tri topologique. La complexité de l'algorithme (O(m)) est plus faible que celle de l'algorithme de Dijkstra ($O(n^2)$).
Les conditions d'utilisation de l'algorithme de Bellman sont satisfaites:

- Graphe sans circuit
- Poids des arcs positifs

Le dernier maillon de la liste chaînée renvoyée par l'algorithme de Bellmann permet d'obtenir le coût total des livraisons.

5. Algorithmes d'analyses des résultats:

Deux autres algorithme sont utilisés pour l'analyse des résultat issus de l'algorithme de Bellman. Avant de les décrire, nous allons expliquer comment interpréter les résultat.

Schéma de la transformation de nos données à partir de l'exemple précédent:

$$\begin{array}{c} \\ D \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{cccccc} D & a & b & c & d & e \end{array} \\ \begin{pmatrix} - & 20 & 25 & 30 & 40 & 35 \\ 20 & - & 10 & 15 & 40 & 50 \\ 25 & 10 & - & 30 & 40 & 35 \\ 30 & 15 & 30 & - & 25 & 30 \\ 40 & 40 & 40 & 25 & - & 15 \\ 35 & 50 & 35 & 30 & 15 & - \end{pmatrix} \end{array}$$

TOUR GEANT

Tableau d'entier (suite de client)				
1	2	3	4	5

SPLIT

Tableau de de Liste_Entier_Cout H (sommet,cout),case noire=espace mémoire non alloué			
H[0]	(1,40)	(2,55)	
H[1]	(2,50)	(3,85)	(4,120)
H[2]	(3,60)	(4,95)	
H[3]	(4,80)	(5,90)	
H[4]	(5,70)		

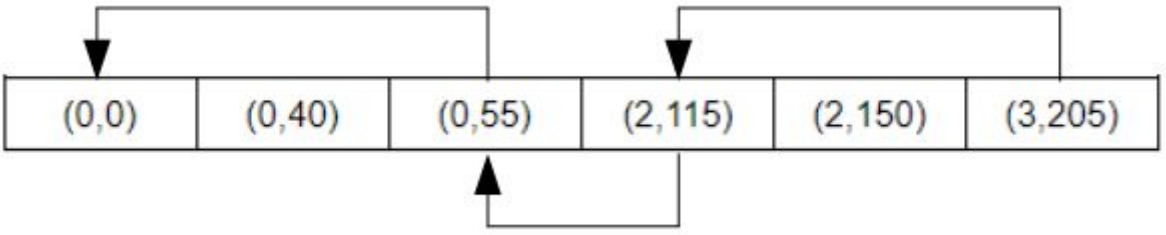
BELLMAN

Liste Entier/Cout (sommet,cout)					
(0,0)	(0,40)	(0,55)	(2,115)	(2,150)	(3,205)

Bellman nous fournit donc l'ensemble des plus courts chemins pour aller à chaque sommet. Ce qui nous intéresse, donc, c'est le plus court chemin pour aller au dernier maillon.
C'est donc avec les 2 algorithmes suivant que l'on va résoudre ce problème.

L'algorithme trajet_camion

Cet algorithme prend en paramètre la liste Entier/Cout des plus courts chemins.
Ainsi pour trouver le plus court chemin entre l'étape de livraison 0 et la dernière il faut parcourir la liste en sens inverse.
Par exemple le plus court chemin pour aller à l'étape 5, est en passant par l'étape3. Le plus court chemin pour aller à l'étape 3,est l'étape 2… Et ainsi de suite jusqu'à arrive à l'étape 0.

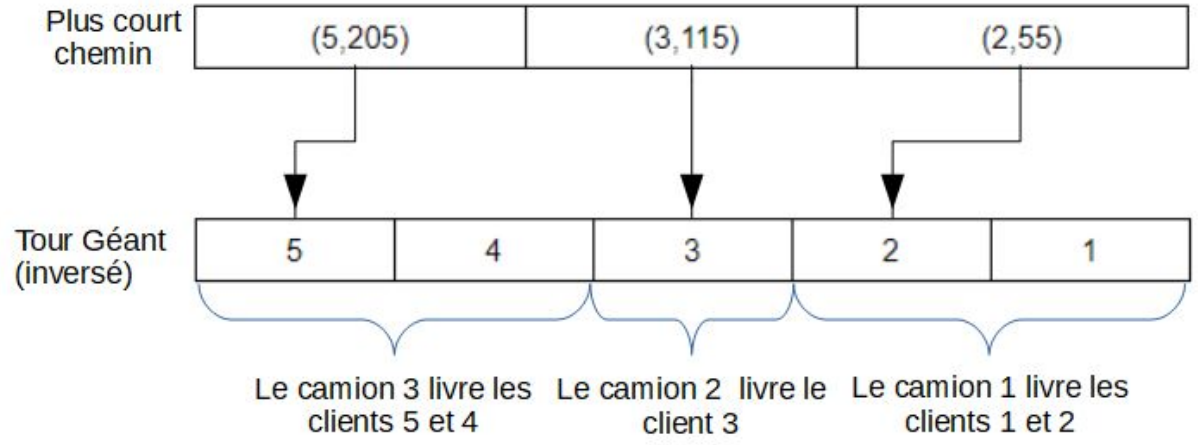


On obtient donc à la suite de cet algorithmes la liste chaînée suivante:

(5,205)	(3,115)	(2,55)

Elle se lit ainsi:
Il y a besoin de 3 camions pour satisfaire tous les clients(car 3 maillons).
Le premier camion part du dépôt et le dernier client de sa tournée est le 2, le coup de sa tournée est de 55-0=55.
Le second camion part du dépôt et le dernier client de sa tournée est le 3, le coup de sa tournée est de 115-55=60.
Le troisième camion part du dépôt et le dernier client de sa tournée est le 5, le coup de sa tournée est de 205-115=90.

Il ne reste plus qu'à déterminer l'ensemble des clients livrés par chaque camion. Il faut donc faire la correspondance entre les derniers clients livré et le tour géant. Le premier camion va donc livrer les client compris dans les indices [2,0[du tour géant. Le second [3,2[et le troisième [5,3[.
L'algorithme client_camion permet de faire cette correspondance, il prend en paramètre le tour géant et le plus court chemin de l'étape 0 à la dernière étape de livraison.



Ainsi à la fin de cet algorithme nous obtenons le tableau de liste suivant:

camion[0]	1	2
camion[1]	3	
camion[2]	4	5

Il ne reste plus qu'à afficher les résultats.

6. Pseudo code

Algorithme du tour géant :

```
Fonction Tour_Geant(n, graphe, C) : Tableau [1,...,n] d'entiers
        Données
                n : entier { nombre de clients}
                graphe : matrice[1,..,n+1][1,..,n+1] de réels
                C : entier { premier client livré }
        Locales :
                suivant, Current : entier
                cout : Réel
                cmp:Entier
                Mark : vecteur [1,..,n ] de booléen {marquage des clients}
        Résultat :
                T : vecteur [1,..,n ] d'entiers { ordre des Clients }


        {Initialisation }

        Pour i de 1 à n Faire
        Mark[i]<- 0 { Le sommet i  n'a pas encore été exploré }
        FinPour
        T[1]<-C
        T[n]<-1 {le dernier sommet du résultat est initialisé à -1}
        Mark[C] <- 1

        cmp <- 2
        cout<- -1

        Current <- T[1]

        Tant que (T[n] = -1 ) Faire
                Pour i de 1 à n Faire
                        Si (Mark[i] == 0 ) et ( i != Current )
                        Si (graphe[Current+1][i+1] < cout ) ou ( cout = -1 )
                                        cout <- graphe[Current+1][i+1]
                                        suivant <- i
                        FinSi
                        FinSi
                FinPour
        T[cmp]<-suivant
        Mark[suivant]=1
        Current<-suivant
        cmp <- cmp +1
        cout <- -1
        FinTantQue
        Renvoyer T
FinFonction
```

Algorithme SPLIT :

```
type Liste_Entier_Cout= structure
                tete : Maillon_Entier_Cout*
                nbelem: Entier
        fin
type Maillon_Entier_Cout= structure
                valeur : entier
                suivant : Maillon_Entier_Cout *
                cout : Réel
        fin

fonction SPLIT(T, Q, n, graphe, q) : Liste_Entier_Cout
        Données :
                T : Vecteur [1,..,n] d'entiers
                Q, n  : Entiers
                graphe : matrice[1,..,n+1][1,..,n+1] de réels
                q : vecteur q[1,..,n] d'entiers
        Locales :
                cost, load, i, j : entiers

        Résultat :
                H : vecteur [1,..,n] de Liste_Entier_Cout

        Pour i de 1 à n  Faire
                j <- i
                load <- 0 { nouveau véhicule affrété }
        Faire load <- load + q[T[j]]
                Si ( i = j ) alors
                        cost <- graphe[ 1 ][ T[i] ] + graphe[ T[i] ][ 1 ]
                Sinon
                        cost <- cost - graphe[ T[j-1] ][ 1 ] + graphe[ T[j-1] ][ T[j] ] + graphe[ T[j] ][ 1 ]
                FinSi
                Si ( load ≤ Q ) alors
                        ajout_en_queue_entier_cout(H[i-1], j, cost)
                FinSi
                j <- j + 1
        Jusque ( j > n ) ou ( load ≥ Q )
        FinPour
        Renvoyer H
FinFonction
```

Algorithme de Bellman :

```
action Bellman(*H,*l,n) : Liste_Entier_Cout
    Données :
        H : Vecteur  dynamique de Liste_Entier_Cout
        n : entier ( nombre de clients )
    Locales :
        père : Vecteur dynamique (taille=nombre de client+1)  d'entiers
        Cout : Vecteur dynamique (taille=nombre de client+1) de flottants
        inter : double
        M : Maillon_entier_cout*
        i : entier
    Résultat :
        L : Liste_Entier_Cout

    "Initialisation"

    Pour i allant de 1 à n Faire
        Cout[i] <- +infini
        père[i] <- -1
    Fin Pour

    allocation_mémoire(Cout,n) #Alloue la mémoire nécessaire à Cout selon n
    allocation_mémoire(père,n)

    "Résolution"

    Pour i allant de 1 à n Faire
        M <- H[i].tete

        Tant que M ≠ NIL Faire
            inter <- Cout[i]+M->cout

            Si (inter<Cout[M->value] ou Cout[M->value]=+infini) alors[
                Cout[M->value] <- inter
                pere[M->value] <- i
            Fin SI
        M <- M->suivant
        Fin Tant que
    Fin Pour

    Pour i allant de 1 à n Faire
        ajouter_en_queue(L,Cout[i],pere[i])
    Fin Pour

    Fin Action
```

La fonction cout étant une fonction renvoyant le coût du chemin entre deux sommets passés en paramètres.

La fonction recherche_predecesseurs est une fonction vue en cours qui recherche un entier dans une liste chaînée d'entiers. Ici dans Liste_Entier_Cout.

Fonction principale du programme :

```
action main():
    Locales:
        Q,n,c: Entier
        graphe : matrice [n+1][n+1] de flottants
        q: vecteur [1,..,n ] d'entiers {demandes client}
        T : vecteur dynamique d'entiers {Tour géant}
        H : vecteur [1,..,n] de Liste_Entier_Cout {sous-graphe H}
        L : Liste_Entier_Cout {résultat de l'algorithme de Bellman}


    lecture_client(n);{Lit le nombre de clients}
    lecture_quantite(Q); { Lit la capacité maximum des camions}
    lecture_demandes(n,q); { Lit la demande pour chaque client }
    matrice_adjacence=lecture_matrice(n); { construit la matrice de coûts entre chaque sommet }
    affichage_info(n,Q,q,matrice_adjacence);

    c =  sommet_depart(graphe) {permet de renvoyer le sommet de départ pour la fonction tour_geant, dans notre cas, nous choisissons le sommet le plus proche du dépot}

    T = tour_geant(n,graphe,c)
    H = SPLIT(T,Q,n,graphe,q)
    init_Liste_Entier_Cout(L)
    Bellman(H,L,n)

    Afficher("Nombre de trajets ( camions ) :“)
    Afficher(camion->nbelem)
    Afficher("Cout total :”)
    Afficher (L->queue->cout)
FinAction
```

7. Mode d'emploi

Pour compiler le projet, nous avons écrit un script permettant d'automatiser la compilation. Pour l'exécuter, il suffit de se placer dans un terminal à la racine du dossier et de rentrer la commande ./script.sh. Si la compilation se passe correctement, la fenêtre devrait afficher les messages suivants :

```
sdescamp@clodion07 ~/Desktop/projetGC $ ./script.sh
lundi 27 mai 2019, 16:46:11 (UTC+0200)
Compilation...
Fini
```

Remarque : Il est possible que vous n'ayez pas les permissions pour exécuter le script. Pour y remédier, entrez la commande suivante dans le même terminal : chmod 777 script.sh.

Une fois le projet compilé, vous disposez d'un exécutable nommé main.

Notre programme étant configuré pour lire une entrée correspondant à un schéma précis qui nous a été imposé, il est nécessaire lors de l'exécution de passer un fichier en entrée. Pour cela, on utilise la commande suivante : ./main < monfichier.exemple .
Voici un exemple :

```
sdescamp@clodion07 ~/Desktop/projetGC $ ./main < exemple.dat
```

Remarque : Il est possible de rediriger la sortie du programme vers un fichier texte par exemple. Pour cela, exécutez la commande suivante :
./main < monfichier.exemple > masortie.txt
De cette façon, vous pourrez sauvegarder la sortie du programme.

À la sortie de notre programme, nous obtenons les informations suivantes:

- Le nombre de clients
- La capacité des camions
- Le nombre de trajets qui seront effectués (nombre de camions)
- Le coût total des livraisons
- Les chemins parcourus par chaque camion.

8. Exemple d'utilisation

Voici un exemple d'utilisation de notre programme :
Nous choisirons ici de passer le fichier exemple.dat en entrée de notre programme main (comme montré dans la partie précédente).

```
tfirmin@gedeon12 ~/Projet SD-GC/ProjetGC $ ./script.sh
jeudi 6 juin 2019, 17:43:08 (UTC+0200)
Compilation...
Fini

tfirmin@gedeon12 ~/Projet SD-GC/ProjetGC $ ./main < cvrp_100_1_r.dat
Nombre de clients: 100
Capacité des camions: 200
Nombre de trajets ( camions ) : 9
Cout total : 1014.550000

Camion n°1 => 14 clients
[41 , 4 , 24 , 29 , 65 , 66 , 71 , 35 , 34 , 78 , 50 , 76 , 12 , 26 ]--Cout:197.860000--

Camion n°2 => 7 clients
[13 , 86 , 38 , 43 , 15 , 57 , 2 ]--Cout:107.630000--

Camion n°3 => 12 clients
[87 , 42 , 14 , 44 , 16 , 61 , 98 , 37 , 100 , 91 , 85 , 93 ]--Cout:92.220000--

Camion n°4 => 9 clients
[96 , 99 , 59 , 92 , 97 , 95 , 94 , 6 , 89 ]--Cout:48.520000--

Camion n°5 => 16 clients
[52 , 18 , 83 , 60 , 5 , 84 , 17 , 45 , 8 , 46 , 64 , 63 , 32 , 90 , 10 , 62 ]--Cout:165.970000--

Camion n°6 => 10 clients
[11 , 19 , 49 , 36 , 47 , 48 , 82 , 7 , 88 , 31 ]--Cout:115.190000--

Camion n°7 => 12 clients
[70 , 30 , 20 , 51 , 9 , 81 , 33 , 79 , 3 , 77 , 68 , 80 ]--Cout:96.520000--

Camion n°8 => 12 clients
[54 , 55 , 25 , 67 , 23 , 39 , 56 , 75 , 22 , 74 , 72 , 73 ]--Cout:115.450000--

Camion n°9 => 8 clients
[21 , 40 , 58 , 53 , 28 , 27 , 69 , 1 ]--Cout:75.190000--

100 clients livrés

--Fin--
```

La liste des clients visités par le camion commence par la fin. Par exemple, le camion n°1 visite 14 client. Il finit par le client n°41, en passant par le client n° 4,...,en commençant par le client n°26.
Le tour géant se retrouve en commençant par le dernier camion.

De plus il n'y a aucune fuite mémoire dans notre projet:

```
==22461==
==22461== HEAP SUMMARY:
==22461==     in use at exit: 0 bytes in 0 blocks
==22461==   total heap usage: 1,044 allocs, 1,044 frees, 1,159,228 bytes allocated
==22461==
==22461== All heap blocks were freed -- no leaks are possible
==22461==
==22461== For counts of detected and suppressed errors, rerun with: -v
==22461== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

(résultat de valgrind pour cvrp_100_1_det.dat).

9. Amélioration possible:

Nous aurions pu mieux adapter notre structure pour l'analyse des résultats du plus court chemin entre l'étape 0 et la dernière. En effet lorsque l'on souhaite établir ce PCC, nous parcourons plusieurs fois la liste (exemple: Pour aller à l'étape 5 il faut passer par l'étape 3, donc nous parcourons la liste jusqu'au maillon 3, et ainsi de suite). Pour éviter cela nous aurions pu convertir la liste chainée en 2 tableaux dynamiques. Un tableau contenant l'étape précédente (d'indice i), de l'étape actuelle i. Et un tableau contenant les coûts de chaque étape.

Ainsi la lecture de l'affichage aurait été plus simple. On ne lirait pas du dernier client livré vers le premier, mais du premier client vers le dernier. Cet ordre nous est imposé par la structure que nous avons choisi et son mode de parcours lors de l'affichage (maillon après maillon).

10. Difficultés rencontrées

La principale difficultée rencontrée lors de ce projet était l'indiçage des boucles dans les algorithmes principaux (Bellman, grand tour, split, …). En effet, quelques erreurs dans l'indiçage nous ont parfois causé des erreurs de segmentation par exemple.
Il était également parfois difficile de choisir une représentation en graphe, par exemple pour stocker les coûts des trajets entre les sommets, nous avons remarqué que nous avions une matrice symétrique et qu'il nous suffisait simplement d'utiliser une matrice triangulaire supérieure ou inférieure. Cependant, l'implémentation étant difficile nous avons opté par une simple matrice allouée dynamiquement.
Finalement, l'interprétation du résultat de la fonction Bellman était difficile à mettre en place. En effet, nous souhaitions afficher les trajets effectués par chaque camion. Pour cela, nous devons donc récupérer le graphe en sortie de Bellman et remonter les éléments de la liste en partant de la queue pour chaque camion.
De plus, ayant manqué de rigueur au début du projet, nous n'avons pas commenté nos algorithmes, et il a été difficile de les commenter par la suite.
Finalement, notre plus grande difficulté a été de gérer les fuites de mémoire. En effet, nous avons pensé à vérifier les fuites de mémoire qu'une seule fois le projet terminé. Nous avons par conséquent passé près de deux heures à essayer de les corriger.

11. Ressentis personnel :

Thomas : Le projet était intéressant. Le délais pour rendre le premier rapport était assez court, ce qui ne nous a pas laissé beaucoup de temps pour réfléchir sur le pseudo-code de nos algorithmes. Le projet a permis de faire prendre conscience de la nécessité d'établir de bonnes structures données, adaptées à chaque problématique. Et à apprendre à bien gérer l'espace mémoire que l'on alloue à nos programmes. Le projet m'a aussi permis de mieux comprendre le fonctionnement des différents algorithmes vus en Graphes.

Simon : Malgré le peu de temps qui nous a été accordé pour produire le rapport d'analyse, la précision du sujet nous a aidé à rapidement comprendre la problématique. Ce projet nous a appris qu'il est important d'être très rigoureux dans les codes utilisant des structures et des allocations dynamiques de mémoires afin d'éviter les erreurs de segmentation et les fuites de mémoire. Je pense que le projet ne nous est pas proposé au moment le plus favorable. En effet, cette dernière semaine nous avons eu nos examens ainsi qu'un projet de calcul numérique et un TP de régression linéaire à rendre. La conséquence est que nous avons manqué de temps sur ce projet. Nous aurions probablement pu rendre un code beaucoup plus propre et plus lisible si nous avions eu d'avantage de temps (par exemple une seule fonction pour libérer tout l'espace mémoire dans le main). Ce projet n'en était pas moins intéressant.